Comput Manag Sci (2011) 8:355–370
DOI 10.1007/s10287-010-0125-4

ORIGINAL PAPER

Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems

Jean-Paul Watson · David L. Woodruff

Received: 20 August 2009 / Accepted: 13 July 2010 / Published online: 29 July 2010

Abstract Numerous planning problems can be formulated as multi-stage stochastic programs and many possess key discrete (integer) decision variables in one or more of the stages. Progressive hedging (PH) is a scenario-based decomposition technique that can be leveraged to solve such problems. Originally devised for problems possessing only continuous variables, PH has been successfully applied as a heuristic to solve multi-stage stochastic programs with integer variables. However, a variety of critical issues arise in practice when implementing PH for the discrete case, especially in the context of very difficult or large-scale mixed-integer problems. Failure to address these issues properly results in either non-convergence of the heuristic or unacceptably long run-times. We investigate these issues and describe algorithmic innovations in the context of a broad class of scenario-based resource allocation problem in which decision variables represent resources available at a cost and constraints enforce the need for sufficient combinations of resources. The necessity and efficacy of our techniques is empirically assessed on a two-stage stochastic network flow problem with integer variables in both stages.

Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

J.-P. Watson
Discrete Math and Complex Systems Department, Sandia National Laboratories,
P.O. Box 5800, MS 1318, Albuquerque, NM 87185-1318, USA
e-mail: jwatson@sandia.gov

D. L. Woodruff (✉)
Graduate School of Management, University of California,
Davis, CA 95616-8609, USA
e-mail: dlwoodruff@ucdavis.edu

1 Introduction

Many planning problems are effectively modeled as multi-stage stochastic programs (Birge and Louveaux 1997; Hochreiter and Pflug 2009; Kall and Wallace 1994; Wallace and Ziemba 2005). These problems often possess key discrete decision variables in one or more of the stages, e.g., binary decisions to take certain actions or integer decisions concerning resource acquisitions. When confronted with either a very large or difficult mixed-integer stochastic programming problem for which there exist effective techniques for solving individual scenarios, the Progressive hedging (PH) algorithm proposed by Rockafellar and Wets (1991) can be leveraged. PH, sometimes referred to as a *horizontal* decomposition method because it decomposes stochastic programs by scenarios rather than by time stages, possesses theoretical convergence properties when all decision variables are continuous. In the presence of discrete decision variables, PH can be effectively used as a heuristic (Fan and Liu 2010; Listes and Dekker 2005; Løkketangen and Woodruff 1996). However, both convergence and efficiency are major issues in the design of PH-based heuristics. In this paper, we introduce algorithmic innovations that enable convergence and yield tractable run-times for PH in the discrete case, concentrating specifically on a large class of stochastic mixed-integer programs for resource allocation.

We focus on the situation where the decision maker considers a set of representative scenarios. We are not concerned here with the origin of the scenarios, but in planning applications they are typically generated by a simulation, constructed directly, or are the result of sampling continuous distributions. For an individual scenario s, many problems of practical interest can be cast in the general framework of constrained optimization:

$$\begin{aligned} &\text{minimize} \quad c \cdot x_s \\ &\text{subject to:} \; x_s \in \mathcal{Q}_s \end{aligned}$$

where x_s is a decision vector of length n, c is a cost coefficient vector of length n, and the requirement $x_s \in \mathcal{Q}_s$ expresses the problem constraints, i.e., to ensure x_s is a feasible solution in scenario s. We use the subscript s to emphasize that the specific problem characteristics will depend on the scenario that is actually observed; the set $\mathcal{S}$ denotes the set of possible scenarios.

Prescient decision makers can simply make use of the decision vector x_s^* that is optimal for the scenario s that they somehow know to be the scenario that will be ultimately realized. All real-world decision makers must make a decision even though *a priori* they are not sure which scenario will be ultimately realized. An optimization model must therefore possess some mechanism(s) for dealing with this uncertainty.

For each scenario $s \in \mathcal{S}$, we denote the probability of occurrence by $\Pr(s)$. These probabilities allow us to take into account prior knowledge of the distribution of individual scenarios, or to weight the relative importance of particular scenarios based on problem-specific knowledge. For the resource allocation problems discussed later in this paper, a typical goal is minimize the expected investment cost. In the two-stage case (presented throughout this paper for notational simplicity; PH and our

computational techniques are generally applicable in the multi-stage case, although it is worth commenting that we present here no empirical evidence that our techniques will exhibit similar computational effectiveness in the multi-stage case), this problem can be mathematically described as follows:

$$\begin{aligned} &\text{minimize } (c \cdot x) + \sum_{s \in \mathcal{S}} \Pr(s)(f_s \cdot y_s) \quad \text{(EF)} \\ &\text{subject to: } (x, y_s) \in \mathcal{Q}_s \quad \forall s \in \mathcal{S} \end{aligned}$$

where the use of a common decision vector x (with $x_s = x, \forall s \in \mathcal{S}$) that does not depend on the scenario implicitly implements the *non-anticipativity* constraints that avoid allowing the decisions to depend on the scenario. The y_s variables represent second-stage, scenario-specific decision vectors with associated cost coefficient vectors f_s, which are determined given x and a particular $s \in \mathcal{S}$. Problem (EF) is the well-known extensive form of a two-stage stochastic program (Wallace and Ziemba 2005).

For such an optimization problem, the basic PH algorithm can be stated as follows, taking a penalty factor $\rho > 0$ and a termination threshold ϵ as the input parameters:

1. $k := 0$
2. For all $s \in \mathcal{S}$, $x_s^{(k)} := \operatorname{argmin}_{x, y_s} (c \cdot x + f_s \cdot y_s) : (x, y_s) \in \mathcal{Q}_s$
3. $\bar{x}^{(k)} := \sum_{s \in \mathcal{S}} \Pr(s) x_s^{(k)}$
4. For all $s \in \mathcal{S}$, $w_s^{(k)} := \rho(x_s^{(k)} - \bar{x}^{(k)})$
5. $k := k + 1$
6. For all $s \in \mathcal{S}$,

$$x_s^{(k)} := \operatorname*{argmin}_{x, y_s} \left(c \cdot x + w_s^{(k-1)} x + \rho/2 \left\| x - \bar{x}^{(k-1)} \right\|^2 + f_s \cdot y_s \right) : (x, y_s) \in \mathcal{Q}_s$$

7. $\bar{x}^{(k)} := \sum_{s \in \mathcal{S}} \Pr(s) x_s^{(k)}$
8. For all $s \in \mathcal{S}$, $w_s^{(k)} := w_s^{(k-1)} + \rho \left(x_s^{(k)} - \bar{x}^{(k)} \right)$
9. $g^{(k)} := \sum_{x \in \mathcal{S}} \Pr(s) \left\| x_s^{(k)} - \bar{x}^{(k)} \right\|$
10. If $g^{(k)} < \epsilon$, then go to Step 5. Otherwise, terminate.

PH provably converges, in linear time, when the decision vector x is continuous. However, termination criteria other than convergence of the $x_s^{(k)}$ to a common $\bar{x}$ are often practically desirable to avoid long run-times and mitigate issues associated with numerical tolerances.

Integer constraints on elements of the decision vectors x and y_s render stochastic programming problems non-convex and add considerable difficulty to their solution. A variety of algorithms for solving such stochastic (mixed-)integer programming problems have been proposed (e.g., see van der Vlerk (1996–2009)). For some smaller and comparatively easier problem instances, standard mixed-integer programming (MIP) solvers can be used to directly solve the extensive form of the problem in which all

scenarios $s \in \mathcal{S}$ are considered simultaneously, as in formulation (EF) (Parija et al. 2004). However, for the operations management problems of interest to us, the extensive forms are either too large or too difficult to solve using standard, commercially available MIP solvers (e.g., CPLEX) in practical run-times.

Several algorithms exist for solving specialized classes of stochastic mixed-integer programs, e.g., the integer L-shaped method and others (Birge and Louveaux 1997). Alternatively, PH can serve as a heuristic for the general case, with integer decision variables in any stage, and with no limitation on the number of stages. Further, given a set of representative scenarios, PH is—because the decomposition strategy is scenario-based—a natural and conceptually straightforward heuristic. Although the integer variables increase solution difficulty, they can be used to speed convergence because equality is well-defined (i.e., two integer variable values are considered equal if the absolute value of their difference is less than some tolerance threshold; we use 1e-5, as does CPLEX by default) and easily detected (Løkketangen and Woodruff 1996). An alternative approach is to use PH to solve a variant of the stochastic program in which the integer constraints are relaxed, and then round to achieve integrality upon termination (Listes and Dekker 2005), although for the problems we consider this technique frequently yields poor-quality solutions.

A large class of real-world resource allocation problems can be characterized through the use of integer variables that represent resources of some sort that can be purchased, with per-unit costs given by the non-negative vector c. For such problems, the binding constraints effectively put lower limits on the values of x, perhaps in complicated ways or perhaps as simple as $Ax \geq b$ for matrix A and vector b with non-negative elements and x constrained to be non-negative. This family of problems is often referred to as diet problems (Garille and Gass 2001), which are sometimes generalized to constrain Ax from both sides. In many diet problems, the decision vector x is only constrained from below. We will refer to this type of problem as a *one-sided* diet problem. The problem we address in this paper resides in this class, and we propose and demonstrate various methods for practical deployment and acceleration of PH for solving this class of problem.

The remainder of this paper is organized as follows. In Sect. 2 we describe innovations for PH that allow us to compute values for the penalty parameter ρ that is based on the input data, accelerate convergence, detect non-convergence, and terminate PH based on prospects for further improvement. The necessity and computational effectiveness of these techniques is experimentally assessed in Sect. 3. We conclude in Sect. 4 with a discussion of the impact of our results and directions for further research.

2 PH algorithmic innovations

Our experience in applying PH to large, real-world stochastic mixed-integer programs led us to develop a number of algorithmic enhancements to the basic algorithm, which can be sub-divided into the following three categories: effective ρ value computation (Sect. 2.1), convergence accelerators (Sect. 2.2), and termination criteria (Sect. 2.3). We additionally describe critical techniques for detecting cycling behavior in PH in

Sect. 2.4. We emphasize that these techniques are (as shown empirically in Sect. 3) necessary to achieve practical PH performance in the role as a heuristic for stochastic mixed-integer programs.

2.1 Computing effective ρ values

We first introduce techniques for selecting ρ in proportion to the unit-cost of the associated decision variable and an alternative, mathematically-based heuristic approach. Variable-dependent ρ strategies have not been previously explored in the literature. Early—and even many recent—reported experiments involving PH have used fairly small values of the penalty parameter ρ, with a single scalar used for all variables. For example, Mulvey and Vladimirou (1991) report that the best values of ρ were much less than 1 and that performance was sensitive to the choice of ρ. However, the particular value they obtained is an artifact of data scaling. For example, Listes and Dekker (2005, p. 374) observe that "there is no conclusive theoretical analysis to support a general selection rule for [ρ]". In their experiments, the best results were obtained using ρ values between 50 and 100. Meanwhile, Fan and Liu (2010) reported that for their problems a different range of values of ρ resulted in reasonable convergence rates.

In the context of the resource allocation problems we consider in Sect. 3, examination of the weighted objective formula for PH (Step 6 of the pseudocode presented in Sect. 1) suggests that significantly larger values of ρ may be required to achieve convergence in practical time-frames. We observe that elements $c(i)$ of the cost vector c may range in magnitude from several hundred dollars to several million dollars per decision variable element, which typically represent expensive resources such as vehicles, road routes, spare parts, or other resources. In the case of an expensive element i, an effective ρ value should be close in magnitude to the unit cost $c(i)$. Otherwise, computation of the initial $w_s^{(k)}(i)$ (Step 4 of the pseudocode in Sect. 1) will yield a small fraction of $x(i)$ – whose value represents a quantity, commonly less than 100 units (e.g., in the case of procurement of expensive resources) – and the per-iteration change in the penalty term $w_s^{(k)}(i)x(i)$ will be comparatively small. Slow changes in the penalty terms necessarily yield little movement in $x(i)$, which in turn significantly delays PH convergence. As a corollary, we comment that the optimal ρ value for a given problem need *not* be fixed at a constant value, i.e., the introduction of per-iteration $\rho(i)$ may in fact be more appropriate for some problems, including those examined in Sect. 3.

Based on these observations, we have developed novel and simple methods for determining element-specific $\rho(i)$ values based on problem-specific data. As demonstrated in Sect. 3, the methods result in substantially improved PH performance relative to constant ρ values, and partially alleviate the need for problem-dependent parameter tuning. Variable-specific $\rho(i)$ values are straightforwardly incorporated into the PH algorithm pseudo-code presented in Sect. 1: the scalar-vector product terms with scalar ρ in Steps 4, 6, and 8 are instead interpreted as component-wise multiplication with a ρ vector.

To motivate the method, consider a scalar quantity x for which non-anticipativity must be enforced. Consequently, only a single corresponding w weight multiplier is

required. Suppose that x is constrained to be an integer taking on small values. Suppose further, that at optimality $w = w^*$ is quite large. This can occur, for example, when x is the quantity of an expensive resource and other (perhaps numerous) variables represent lower cost operational decisions. If ρ is small, this situation will result in many iterations required for convergence of PH because at each iteration, w can grow only by the product of two small quantities.

Our objective is to develop a heuristic method of setting ρ that will allow the updates to proceed more quickly to a "good" value w^* of the weight w. For practical reasons, we want the magnitude of w to approach from below in order to minimize oscillation or thrashing. Oscillation can occur when the w values are updated too aggressively or converge from both sides particularly in MIPs because the changes in the value of one integer variable can induce changes in others, which are then reversed if the w multiplier "shoots past" its optimal value. Before proceeding, we note that the motivation for our heuristic is based on separability of the decision variables, although it is not required for use of the method. For clarity of the motivation, we proceed in the context of a single variable and linear objective function.

Consider a single decision variable x with corresponding cost coefficient c; individual problem scenarios are denoted by s. After iteration zero of PH completes, we have an estimate of the optimal value for x, which is $\bar{x}^{(0)}$. If we set a value of ρ that will result in $w = c$, then the proximal term

$$\rho/2 \left\| x_s^{(k-1)} - \bar{x}^{(k-1)} \right\|^2$$

will force the solution to be $\bar{x}^{(0)}$ in the subsequent PH iteration. The value of w is updated by

$$w_s^{(k)} := w_s^{(k-1)} + \rho \left(x_s^{(k-1)} - \bar{x}^{(k-1)} \right)$$

so the value of ρ for a given scenario s resulting in $w = c$ is

$$\rho_s := \frac{c}{|x_s - \bar{x}^{(0)}|}$$

We want the absolute value of all w elements to approach their ultimate value from below to help mitigate thrashing or cycling that can occur when integers change values forcing jumps in the values of other variables, so we use a bound on the denominator and drop the dependence on s. After PH iteration 0, for each variable x we define $x^{\max} = \max_{s \in \mathcal{S}} x_s^{(0)}$ and $x^{min} = \min_{s \in \mathcal{S}} x_s^{(0)}$. Since $(x^{\max} - x^{\min} + 1) > |x_s - \bar{x}|$ we use

$$\rho(i) := \frac{c(i)}{(x^{\max} - x^{\min} + 1)}$$

for variable i, which does not depend on s. This scheme is used for blending all integer variables in our experiments. In contrast, continuous variables can change gradually

without large discrete jumps, so it is not necessary to use such a conservative denominator. Thus, we instead use the following formula to determine $\rho(i)$ for the continuous variables in our experiments:

$$\rho(i) := \frac{c(i)}{\max\left(\left(\sum_{s\in\mathcal{S}} \Pr(s)|x_s^{(0)} - \bar{x}^{(0)}|\right), 1\right)}$$

We denote this heuristic method for selecting per-element $\rho(i)$ by SEP.

The primary advantages of the ρ selection heuristic SEP are its problem-independent nature and the fact that it is parameter-free, eliminating the need for repeated execution of PH in the search for high-quality ρ values. However, there exists a high likelihood that more effective methods exist for any specific problem. We have investigated a number of alternative ρ selection strategies for a range of stochastic mixed-integer resource allocation programs. The best-performing alternatives (including SEP) were all based on the simple observation that the value of $\rho(i)$ should be proportional to element unit cost, as discussed above. We also report on results based on a straightforward yet effective "cost-proportional" method for setting $\rho(i)$. Specifically, we set $\rho(i)$ equal to a multiplier $k > 0$ of the element unit cost $c(i)$. The method is denoted by CP$(\cdot)$, where CP stands for *cost-proportional* and the argument gives the cost multiplier. Finally, as a control measure, we consider the performance of PH using various fixed, global values of ρ. We denote the corresponding method by FX$(\cdot)$, where the FX stands for *fixed* and the argument gives the sole value of ρ.

2.2 Accelerating convergence

Although PH may eventually force agreement among the decision variable vectors x_s to a common vector x, in practice the number of iterations required is frequently impractical for complex, non-convex stochastic integer programming problems.

The following three acceleration methods are designed for one-sided constraints, such as when the problem for each scenario is to minimize $c \cdot x$ subject to $Ax \geq b$ with $x \geq 0$ where the elements of vectors c and b and the matrix A are all non-negative. For problems where the constraints effectively limit x from both sides, these methods may result in PH encountering infeasible scenario sub-problems even though the problem is ultimately feasible. For one-sided resource allocation problems, as we will demonstrate, the methods are however quite effective.

A detailed analysis of PH algorithm behavior on the problems considered in Sect. 3 in addition to other problems indicates that individual decision variables $x_s(i)$ frequently converge to specific, fixed values z for all $s \in \mathcal{S}$ in early PH iterations. Further, despite interactions among the $x_s(i)$ for any particular scenario s, the value z frequently fails to change in subsequent PH iterations. Such variable "fixing" behaviors lead to a potentially powerful, albeit obvious, heuristic: once $x_s^{(k)}(i) = x(i)$ for all $s \in \mathcal{S}$ at a particular PH iteration k, fix $x_s^{(l)}(i) = z$ for all subsequent iterations $l > k$. For both continuous and discrete decision variables, a small tolerance threshold for equality must be imposed; the need in the case of continuous variables is inherent,

while for discrete variables the need is driven by the inexact nature of the LP solvers underpinning the MIP branch-and-bound process. In our experiments, we use a tolerance equal to 1e-5, identical to the tolerance used to determine integer equality (as described in Sect. 1). The precise threshold value is not critical, as the main goal is to ensure that the product of the tolerance and the objective cost coefficient of the variable being fixed is insignificant relative to the overall solution cost. As shown in Sect. 3, variable fixing can yield substantial reductions in solution times by accelerating (through variable elimination) the solution times for individual scenario sub-problems, at the expense of slight reductions in solution quality.

In applying this heuristic, we first introduce a *lag* parameter $\mu \in \{0, 1, \ldots\}$. Consider a given PH iteration k. We fix $x_s^{(k)}(i)$ for all subsequent iterations $l > k$ once $x_s^{(m)}(i) = z$ for all $s \in \mathcal{S}$ and $m \in \{k - \mu|\mathcal{S}|, \ldots, k\}$, such that $m \geq \mu|\mathcal{S}|$. In other words, we fix decision variables once their value has stabilized to a fixed z over the last $\mu|\mathcal{S}|$ PH iterations. Low values of μ yield immediate or near-immediate variable fixing; larger values of μ can respond to the empirically less common event that the value of z may in fact vary over moderate time horizons, i.e., it may become "undone" due to the influence of competing decision variables. The multiplicative factor $|\mathcal{S}|$ accounts for the observation that the number of PH iterations required for convergence grows with the total number of scenarios under consideration.

This idea can be taken further by fixing values for decision variables that have not yet converged as a means of quickly forcing termination of the algorithm, which we refer to as *slamming*. Consider a situation in which it has been determined that the individual scenario solutions $x_s^{(k)}$ are "sufficiently" converged, i.e., they are very nearly homogeneous in both the values of the decision vectors $x_s^{(k)}$ and the scenario costs $c \cdot x_s^{(k)}$. The basic PH algorithm can take very large number of iterations to resolve the remaining discrepancies, despite minimal impact on the final solution quality. One alternative, reported in the literature (Listes and Dekker 2005; Løkketangen and Woodruff 1996), is to solve a variant of the extensive form in which all currently-converged decision variables are fixed to their common value. Another alternative, explored here, is to force absolute PH convergence via aggressive variable fixing.

Once the variables have converged sufficiently (the specific criteria are described subsequently in Sect. 2.3), we first set the lag $\mu = 0$, independent of its current value. Then, every 2 subsequent iterations we identify the free decision variable $x(i)$ for which the total cost $c(i) \cdot \max_{s \in \mathcal{S}} x_s(i)$ is minimal. We then fix $x(i) = \max_{s \in \mathcal{S}} x_s(i)$. In the case of one-sided resource allocation problems, feasibility of the scenario sub-problems is necessarily not lost via such a maximum-value scheme. Clearly, this scheme is guaranteed to force termination. We have investigated performance using various periods between variable fixings, but performance is not sensitive to this choice.

2.3 Termination criteria

In practice, PH empirically yields large reductions in $||x_s - x_{s'}||$ for $s, s' \in \mathcal{S}$ in early iterations, while the remaining and majority of iterations serve in a fine-tuning role to drive the already small differences in $||x_s - x_{s'}||$ to 0. To detect near-convergence in the

solution vectors x_s, $s \in \mathcal{S}$, we first define the normalized average per-scenario deviation from the "average" solution as $td = \left(\sum_{i,s:\overline{x}(i)>0} \frac{|x_s(i)-\overline{x}(i)|}{\overline{x}(i)}\right)/|\mathcal{S}|$, where $\overline{x}(i)$ represents the average of $x_s(i)$ over all $s \in \mathcal{S}$. We then can invoke variable slamming to quickly force PH convergence once td drops below some parametric threshold λ_t. The value of λ_t places a threshold on the degree of heterogeneity allowed in the set of solutions x_s.

Such a termination criterion assumes that small differences in $||x_s - x_{s'}||$ are correlated with small differences in the costs $||c \cdot x_s - c \cdot x_{s'}||$. In practice, however, this is often *not* the case. For example, there can exist "holdout" scenarios that require more high-cost resources than other scenarios to achieve feasibility. Consequently, the value of td may in fact be very small, while the discrepancy in overall costs may be quite large. To protect against such situations, we additionally consider a termination criterion based on the variability of solution quality in any given PH iteration k. For one-sided resource allocation problems, upper bounds on the $x(i)$ are easily obtained, e.g., by considering the total number of a resource i that could ever be used in a given scenario $s \in \mathcal{S}$. At an arbitrary PH iteration k, consider the solutions x_s for all scenarios $s \in \mathcal{S}$. Let $x^{\max}$ denote the decision vector whose elements represent the maximal value appearing in any solution x_s, i.e., $x^{\max}(i) = \max_{s\in\mathcal{S}}(x_s^{(k)}(i))$. The element-wise minimum vector $x^{\min}$ is defined analogously. Clearly, $x^{\max}$ is a feasible (albeit likely suboptimal) solution to all scenarios $s \in \mathcal{S}$, while $x^{\min}$ may be infeasible for some scenarios. Finally, let $qd = (c \cdot x^{\max}/c \cdot x^{\min}) * 100$. We can then terminate PH iterations once qd drops below a parameterized threshold value λ_q, e.g., where $\lambda_q = 1\%$.

In our PH implementations, we invoke variable slamming (as described above in Sect. 2.2) once *both* $td \leq \lambda_t$ and $qd \leq \lambda_q$ after a PH iteration k.

2.4 Detecting cyclic behavior

Finally, we note that for all types of stochastic mixed-integer programs, there is a risk of non-convergence of the PH algorithm—even given the computational techniques we have described above. In the experiments discussed in Sect. 3, cycling behavior is detected at least once in roughly one tenth of all algorithmic trials. Consequently, cycle detection and avoidance mechanisms are required to force eventual convergence of the PH algorithm in the mixed-integer case. To detect cycles, we chose to focus on repeated occurrences of $w_s(i)$ vectors, i.e., the set of scenario weights associated with a decision variable $x(i)$. We consider $w_s(i)$ vectors instead of $x_s(i)$ vectors or even $\overline{x}(i)$ because in the integer case both $x_s(i)$ and $\overline{x}(i)$ can stagnate for long periods (yet remain non-converged), complicating the detection logic. In contrast, the $w_s(i)$ vectors are continually changing – assuming the associated decision variable has not converged, making detection of potential cycles straightforward.

For computational efficiency, we use a simple hashing scheme drawn from Woodruff and Zemel (1993) to detect potential cycles in a variable $x(i)$. When PH is initialized, an integer hash weight z_s for each scenario $s \in \mathcal{S}$ is drawn from a pseudo-random distribution, e.g., using a built-in random number generator (AMPL's Irand224 in our

case). The corresponding hash value $h(i)$ is given by $\sum_{s \in \mathcal{S}} z_s w_s(i)$. Cycle detection is then performed by comparing the $h(i)$ across different iterations of PH. This scheme operates identically for both continuous and discrete variables. Note that hash-based cycle detection is a heuristic, used to minimize impact on run-time. If equal hash values are detected, they are interpreted as evidence for a *potential* cycle; various combinations of the $w_s(i)$, $x_s(i)$, and $\overline{x}(i)$ histories can then be used to verify that a cycle has actually occurred and needs to be broken. In the case of continuous variables, the $h(i)$ are also continuous, requiring the use of a tolerance to detect "equality". In our experiments, we use the tolerance associated with testing equality of integers, as discussed in Sect. 1.

Once a cycle in the weight vectors associated with any decision variable $x(i)$ is detected, the value of $x(i)$ is immediately fixed to $\max_{s \in \mathcal{S}} x_s(i)$; feasibility is again ensured in the case of one-sided resource allocation problems. In practice, few variables are fixed in such a fashion, yielding minimal impact on final solution quality while assuring termination.

3 Experimental analysis of PH performance

We now describe an empirical performance analysis of the various PH algorithmic techniques proposed in Sect. 2. To provide an easily reproduced laboratory example for demonstrating and testing our proposed methods, we consider a basic network flow model with stochastic and resource allocation aspects. The problem formulation is provided in Sect. 3.1. Our problem instance generation scheme is discussed in Sect. 3.2, in addition to our experimental methodology. Computational results are then detailed in Sect. 3.3.

3.1 Stochastic programming formulation

A flow network is defined over a node set $\mathcal{V}$ and an arc set $\mathcal{A}$, $\mathcal{A} \subseteq \mathcal{V} \times \mathcal{V}$. The network structure is static across a set of scenarios $\mathcal{S}$. For each scenario $s \in \mathcal{S}$, a quantity $D_{kl}(s)$ must be shipped between each pair of nodes $(k, l) \in \mathcal{V} \times \mathcal{V}$, $k \neq l$. The first stage decision variables include continuous arc capacities $x(a)$, $a \in \mathcal{A}$, and binary indicators of arc availability $b^0(a)$, for which the cost parameters $c(a)$ and $F^0(a)$ are respectively defined. Additional resources for each arc $a \in \mathcal{A}$ must be deployed after the demand becomes known in order to activate the arc. The use of these resources is defined by the second stage decision variables $b(a, s)$ at a corresponding cost of $F(a, s)$, introduced for each $a \in \mathcal{A}$ and $s \in \mathcal{S}$. The optimization objective is then to minimize the sum of the first stage fixed plus marginal costs and the expected second-stage costs.

Continuous variables $y_{kl}(a, s)$ are introduced to represent the flow from node k to node l passing through arc $a \in \mathcal{A}$ in scenario $s \in \mathcal{S}$. Assuming the availability of a large constant M (e.g., the sum of all demands), the problem formulation, which we denote (SNF), is given as:

$$\text{minimize} \quad \sum_{a\in\mathcal{A}}\left[c(a)x(a) + F^0(a)b^0(a) + \sum_{s\in\mathcal{S}}\Pr(s)F(a,s)b(a,s)\right] \qquad \text{(SNF)}$$

$$\text{subject to:} \quad \sum_{a\in\mathcal{A}^+(v)} y_{kl}(a,s) - \sum_{a\in\mathcal{A}^-(v)} y_{kl}(a,s) = \begin{cases} -D_{kl}(s) & \text{if } v = k \\ D_{kl}(s) & \text{if } v = l \\ 0 & \text{otherwise,} \end{cases}$$

$$\forall v,k,l \in \mathcal{V}, s \in \mathcal{S}$$

$$x(a) \geq \sum_{k,l\in\mathcal{V}} y_{kl}(a,s) \qquad \forall a \in \mathcal{A},\ s \in \mathcal{S}$$

$$y_{kl}(a,s)) \leq Mb^0(a) \qquad \forall k,l \in \mathcal{V}, a \in \mathcal{A}$$

$$y_{kl}(a,s) \leq Mb(a,s) \qquad \forall k,l \in \mathcal{V}, a \in \mathcal{A}, s \in \mathcal{S}$$

$$b^0(a) \in \{0,1\} \qquad \forall a \in \mathcal{A}$$

$$b(a,s) \in \{0,1\} \qquad \forall a \in \mathcal{A}, s \in \mathcal{S}$$

$$y_{kl} \geq 0 \qquad \forall k,l \in \mathcal{V}, k \neq l$$

$$x(a) \geq 0 \qquad \forall a \in \mathcal{A}$$

where the notations $\mathcal{A}^+(v)$ and $\mathcal{A}^-(v)$ respectively indicate the set of arcs into and out of node $v \in \mathcal{V}$. The probability of $s \in \mathcal{S}$ being realized is denoted by $\Pr(s)$. The (SNF) formulation represents the extensive form of the optimization problem; the corresponding single-scenario quadratic MIPs (due to the weighted penalty terms) for use in PH are straightforwardly derived.

3.2 Test problems and experimental methodology

The computational experiments reported in Sect. 3.3 were performed using networks consisting of 10 nodes, consecutively labeled from 1 to 10. The static arc set is given by the following node pairs, each representing a bi-directional arc: (1,2), (1,3), (2,3), (2,4), (2,5), (3,4), (4,5), (5,6), (6,7), (6,8), (7,8), (7,9), (7,10), (8,9), and (9,10). This particular instance is an extension of the 5-node instance introduced in Ruszczyński (2002); our general stochastic network formulation is taken directly from Ruszczyński (2002).

To shorten the exposition and ease reproduction of our results, we use simple, arbitrary formulas to establish parameter values for the instance data. For each arc $a \in \mathcal{A}$ connecting nodes indexed by i and j, we let $c(a) = 100 + 10|i - j| + (ij \mod 10)$. As in Ruszczyński (2002), the $D_{kl}(s)$ for a specific scenario $s \in \mathcal{S}$ are given by $D_{kl}(s) = 0.1D(s) + \epsilon_{kl}$, where $D(s)$ represents the aggregate flow volume for scenario s, sampled from a normal distribution $\mathcal{N}(30, 5)$; the ϵ_{kl} are independent normally distributed variables with mean 0 and standard deviation 0.25. We let $F^0(a) = 10c(a)$ and $F(a,s) = U(s)c(a)$ where $U(s)$ is drawn from a distribution that assigns equal likelihood to the integers 5, 10, and 15. We additionally require all arcs to be bi-directional and the capacities are assumed to be symmetric across an arc, i.e., $a_1 = (i,j) \wedge a_2 = (j,i) \Rightarrow (x(a_1) = x(a_2) \wedge b^0(a_1) = b^0(a_2))$. For testing purposes, we generated 10 and 50-scenario problem sets, each containing 5 instances apiece with

$\Pr(s) = 1/|\mathcal{S}|$. While small in terms of both network scale and $|\mathcal{S}|$, these instances pose significant computational challenges to both our PH variants and commercial MIP solvers, due to the presence of integer variables in both the first and second decision stages.

All models and data were expressed using the AMPL modeling language (http://www.ampl.com; Fourer et al. 2003); all data files are freely available from the authors. CPLEX 10.1 was used to solve both the extensive forms and the PH scenario sub-problems. The PH algorithms were also written in AMPL, to facilitate the required access to algebraic model components. Default parameter settings for CPLEX were used, as exploratory manipulation of heuristic tuning parameters (e.g., local branching) failed to noticeably improve performance.

To establish a performance baseline and to bound final solution quality, we allocated two days (2,880 min) of run-time to ILOG's (http://www.ilog.com.) CPLEX 10.1 mixed-integer programming solver to the extensive form (SNF) of each problem instance, with the best feasible incumbent solution recorded upon termination. A primary experimental objective is to demonstrate the relative benefit of variable versus fixed ρ selection strategies. Consequently, we consider the following ρ selection strategies for PH: CP(1.0), SEP, FX(250), and FX(1000). The parameters for the two fixed ρ strategies—250 and 1000—are respectively chosen based on the approximate mean values of the (SNF) cost parameters $c(a)$ and $F^0(a)$. A secondary experimental objective is to assess the impact of the fix lag parameter μ on PH performance, which we vary among $\{0, 1, 2\}$. To additionally examine interaction effects between the ρ selection strategies and the fix lag μ, we performed a fully crossed experiment, yielding 12 runs for each instance.

For each individual PH run, we set the termination criteria parameters λ_q and λ_t equal to 0.01% and 0.0001, respectively. Individual PH scenarios are also solved with CPLEX 10.1, leveraging the quadratic solver engine. To accelerate scenario solves in early PH iterations (Wets 1989)—which are empirically far more costly than those for later iterations—we monotonically non-increase the CPLEX *mipgap* parameter as PH progresses, setting the value equal to the minimum of the convergence parameter *td* computed at the end of the previous PH iteration and the *mipgap* parameter value used in the previous PH iteration. With the exceptions noted below, all PH variants are executed until convergence to a common x is achieved through the use of variable slamming or other mechanisms described in Sect. 2. At each PH iteration, a maximal-cost solution satisfying all scenarios can be formed by taking the element-wise maximum across the decision variables for all $s \in \mathcal{S}$; feasibility is guaranteed due to the one-sided nature of the resource constraints, and the cost is given as:

$$\sum_{a \in \mathcal{A}} \left[c(a) x^{\max}(a) + F^0(a) b^{0\max}(a) + \sum_{s \in \mathcal{S}} \Pr(s) F(a, s) b(a, s) \right] \tag{1}$$

The best solution found in any PH iteration is recorded, in addition to the overall run-time, rounded to the nearest half-minute increment. All runs are executed on a 64-bit AMD Opteron 2.2 GHz workstation, with 64GB of RAM running Linux 2.6.

Table 1 Small instances: performance results for PH runs on the five 10-scenario stochastic network flow instances

PH algorithm	Avg. % improvement vs. 2,800 min baseline	Average run-time	Average PH iterations
$\mu = 0$, FX(1000)	−14.00	7.6	20.4
$\mu = 0$, FX(250)	−4.75	17.5	63.8
$\mu = 0$, CP(1.0)	−1.81	9.7	41
$\mu = 0$, SEP	**−1.27**	18.9	86.6
$\mu = 1$, FX(1000)	−10.87	75.6	23.4
$\mu = 1$, FX(250)	−2.848	85.7	117
$\mu = 1$, CP(1.0)	−0.72	51.5	61.6
$\mu = 1$, SEP	**−0.68**	61.6	152.8
$\mu = 2$, FX(1000)	−10.21	141.4	23.6
$\mu = 2$, FX(250)	−1.57	129.4	96
$\mu = 2$, CP(1.0)	**−0.73**	73.0	68
$\mu = 2$, SEP	−0.85	321.8	321.8

Performance is relative to the best solution found by CPLEX 10.1 for the extensive form within a time limit of 2800 minutes, and is averaged over the five instances. Average run-times are reported in minutes, rounded to the nearest half-minute increment. For a given μ, the PH configuration yielding the highest-quality performance relative to CPLEX baseline is bold-faced

3.3 Results

We first consider the results for the 10-scenario instances. Despite their small size, the extensive forms of these instances are difficult for CPLEX 10.1. In no case could CPLEX prove optimality within the two day limit, instead yielding a final optimality gap between 1.32 and 2.15%; improvements in the incumbent were observed throughout the runs. To assess the performance of PH, we compute the percentage improvement of each algorithm over the best CPLEX incumbent; negative numbers indicate PH was outperformed by CPLEX. The average results across the five instances are reported in Table 1, in addition to the average run-time and number of PH iterations.

Analyzing the PH results, we observe that the fixed ρ strategies significantly underperform the variable ρ strategies, controlling for the fix lag μ. Runs with FX(1000) yielded the worst performance in terms of solution quality, which is consistent with the overweighting of ρ values (relative to the mean $c(a)$) associated with arc capacities $x(a)$. Intuitively, we expect lower ρ values to yield improved solution quality at the expense of increased run-times. This is observed in runs with FX(250). However, the FX(250) strategy consistently underperforms both the CP(1.0) and SEP variable ρ strategies in terms of solution quality. Further, the CP(1.0) requires less run-time in all cases. The mathematically motivated SEP variable ρ strategy outperforms the CP(1.0), except when $\mu = 2$. Independent of the ρ selection strategy, increases in μ marginally improve final solution quality, but at the expense of significant growth in run-times. While we cannot rule out fixed values of ρ that may outperform our variable ρ strategies, we observe that no tuning was performed for the variable ρ strategies (which

Table 2 Larger instances: performance results for PH runs on the five 50-scenario stochastic network flow instances

PH algorithm	Avg. % improvement vs. 2,800 minute baseline	Average run-time	Average PH iterations
$\mu = 0$, FX(250)	6.06	305	186
$\mu = 0$, FX(1,000)	−3.58	66	163
$\mu = 0$, CP(1.0)	8.16	274	99
$\mu = 0$, SEP	6.92	1,337	485

Performance is relative to the best solution found by CPLEX 10.1 for the extensive form within a time limit of 2,800 min, and is averaged over the five instances. Average run-times are reported in minutes, rounded to the nearest half-minute increment

would mitigate the run-time advantage of fixed ρ strategies), and that the fixed ρ values selected were reasonable and based on general instance family data.

While none of the variable ρ PH configurations outperform the CPLEX baseline in terms of final solution quality (on average; a SEP PH configuration identified a better solution on a single instance), this was expected given the small instance size. The PH run-times are significantly less than the 2,800 min allocated to CPLEX. We note that a two-day run-time budget is often unacceptable in operations management environments, where numerous variants of a particular problem are solved in the course of decision-making. With few exceptions (specifically with the SEP heuristic and $\mu = 2$), both the variable and fixed ρ strategies outperform CPLEX on the extensive form of (SNF) given equivalent run-times, and generally achieve nearly equivalent solution quality in a fraction of the total run-time.

Additionally, we observe that even minor increases in μ significantly inflate PH run-times. On runs with $\mu = 5$ (not reported), the PH run-times were consistent with the 2-day limit provided to CPLEX. Finally, variable slamming to force PH convergence is critical to achieving practical run-times; without this mechanism, PH run-times—even with $\mu = 1$—can reach two days or longer.

We next consider results for the five 50-scenario instances, reported in Table 2. Based on our 10-scenario results, we limit the scope of the experiments. PH run-time empirically grows superlinearly with increases in $|\mathcal{S}|$ and μ, necessitating a more limited investigation. Consequently, we limit $\mu = 0$ due to the growth in run-times, caused by the total number of PH iterations and the aggregate CPLEX run-times on individual scenarios. These instances are significantly more difficult than the 10-scenario instances, with CPLEX optimality gaps on the extensive form ranging between 15.85 and 18.82% upon termination. PH run-times with $\mu \geq 1$ typically exceeded two days, further emphasizing the necessity of variable fixing for effective PH performance in the mixed-integer case.

All but the FX(1000) PH configurations outperform the CPLEX extensive form baseline in terms of both run-time and solution quality, with the best performance obtained by CP(1.0). The latter yields solutions over 8% better than the CPLEX baseline in an order-of-magnitude less run-time. Given equal run-times, all PH configurations dominate the CPLEX baseline in terms of solution quality. Both variable ρ strategies outperform the fixed FX(250) strategy, with the CP(1.0) variant obtaining higher-quality

solutions in less run-time. The performance of the FX(250) variant is, however, reasonably competitive with the variable ρ configurations. It is not surprising that specific, fixed ρ values can yield competitive performance on these laboratory instances, given relatively homogeneous resource costs. The parameter-free SEP strategy obtains high-quality solutions, but with longer run-times relative to CP(1.0). However, this advantage is at least partially mitigated by the need to select reasonable parameter values for the CP($\cdot$) approach. Even with our proposed convergence accelerators, the PH run-times for these small instances are non-trivial; without them, these instances are computationally intractable.

4 Conclusions and directions for further research

Numerous planning problems can be formulated as multi-stage stochastic programs and many have important integer valued decision variables in one or more stages. Progressive hedging (PH) is a scenario-based decomposition technique for heuristically addressing such problems. While PH has been successfully applied to a number of problems, a variety of issues arise when implementing PH in practice, especially when dealing with very difficult or large-scale mixed-integer problems. In particular, decisions must be made regarding the value of the penalty parameter ρ, criteria for termination, and techniques for accelerating convergence.

We investigated PH as applied to a class of scenario-based resource allocation problem in which decision variables represent resources available at a cost and constraints enforce needs for sufficient combinations of resources. We described computational experiments that demonstrate the efficacy of PH and the various enhancements that we have introduced to facilitate its practical, real-world application. We have developed and motivated a problem-independent method for computing good values of the main PH parameter, ρ, that depends on problem-specific data. We additionally describe techniques for accelerating convergence, and detecting cycling behavior and damping it as appropriate. Our experiments indicate that our proposed variable-specific ρ selection strategies outperform scalar ρ strategies that are commonly associated with PH implementations reported in the literature. Further, our experiments indicate that such techniques are *necessary* to achieve practical PH run-times in the mixed-integer case.

Some of the enhancements that we introduce exploit the fact that we are solving resource allocation problems with one-sided constraints, specifically bounding the decision variables from below. Our convergence accelerators—which yield lower run-times with limited impact on quality—rely on such one-sided constraints; the same is true of our termination criteria. While this covers a large and important class of resource allocation problems, it remains as future research to extend these PH enhancements or develop analogs for more generally constrained resource allocation problems.

The algorithmic techniques we developed were necessitated and driven by the scale of our test problems, illustrating the broader need for more complex and realistic problems to drive the development of practical stochastic and robust programming solvers. For example, without variable fixing PH requires weeks of computer time for convergence real-world resource allocation problems we have tackled. There is a rich

and growing literature on algorithms for solving stochastic and robust problems, but research opportunities abound in the area of solving large-scale problems with integer variables. For situations where scenario sub-problems are tractable, but the extensive form cannot be addressed directly, PH provides a mechanism to find good solutions.

Future research is needed for acceleration methods when there are two-sided constraints and for finding good ρ values for variables that do not have cost coefficients. Furthermore, it would be useful to conduct computational experiments concerning the tradeoff between sub-problem solution quality and run-time and overall solution quality. Advanced schemes involving increasing subproblem solution quality over the PH iterations may be particularly fruitful, but the complex interactions with acceleration methods requires experimental investigation.

References

Birge JR, Louveaux F (1997) Introduction to stochastic programming. Springer, Berlin

Fan Y, Liu C (2010) Solving stochastic transportation network protection problem using the progressive hedging-based method. Netw Spatial Econ 10(2):193–208

Fourer R, Gay DM, Kernighan BW (2003) AMPL: a modeling language for mathematical programming, 2nd edn. Thompson Learning, Brooks/Cole

Garille SG, Gass SI (2001) Stigler's diet problem revisited. Opera Res 49(1):1–13

Hochreiter R, Pflug GCh (2009) Special issue on computational optimization under uncertainty. Comput Manag Sci 6(2):pp 115–267

Kall P, Wallace SW (1994) Stochastic programming. Wiley, Chichester

Listes O, Dekker R (2005) A scenario aggregation based approach for determining a robust airline fleet composition. Transp Sci 39:367–382

Løkketangen A, Woodruff DL (1996) Progressive hedging and tabu search applied to mixed integer (0,1) multistage stochastic programming. J Heurist 2:111–128

Mulvey JM, Vladimirou H (1991) Applying the progressive hedging algorithm to stochastic generalized networks. Ann Oper Res 31:399–424

Parija GR, Ahmed S, King AJ (2004) On bridging the gap between stochastic integer programming and MIP solver technologies. INFORMS J Comput 16:73–83

Rockafellar RT, Wets RJ-B (1991) Scenarios and policy aggregation in optimization under uncertainty. Math Oper Res 16:119–147

Ruszczyński A (2002) Probabilistic programming with discrete distributions and precedence constrained knapsack polyhedra. Math Program 93:195–215

van der Vlerk MH (1996–2009) Stochastic integer programming bibliography. http://mally.eco.rug.nl/index.html?spbib.html

Wallace SW, Ziemba WT (eds) (2005) Applications of stochastic programming, SIAM. Cambridge University Press, Cambridge

Wets RJ-B (1989) The aggregation principle in scenario analysis and stochastic optimization. In: Wallace SW (ed) Algorithms and model formulations in mathematical programming. Springer, Berlin

Woodruff DL, Zemel E (1993) Hashing vectors for tabu search. Ann Oper Res 41:123–137